exalate

How to Use Exalate to Synchronize Insight Objects

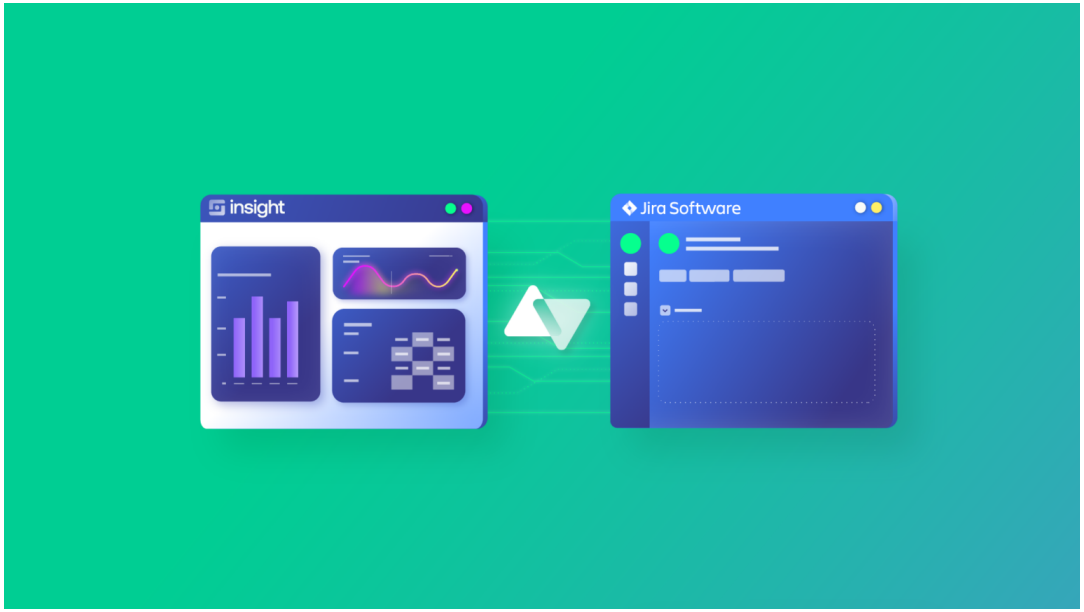


Table of contents

Exalate was designed with the core concept of **flexibility** in mind and that is the feature that allows Exalate to be used as a solution in a variety of integration use cases, including this precise use case of synchronizing Insight Objects.

So in this article, we are going to focus more on Exalate's flexibility and will walk you through how it can be used to synchronize Insight objects. We'll show you:

- How to retrieve object attributes including the following references
- How to search for objects
- How to create new objects

The Scenario

Team Blue is using Jira Service Desk to manage all the details to get Game of Thrones out of the door. **Team Blue** is working with **Team Green** for all costumer-related requests. Team Green is actually an external team using Jira to manage all the requests from multiple customers.

As each customer has their own ways of managing their information, a mapping will be required between the two environments. Furthermore, both companies are using Insight Asset Management for keeping track of their assets.

Note: *This is a purely fictional scenario.*

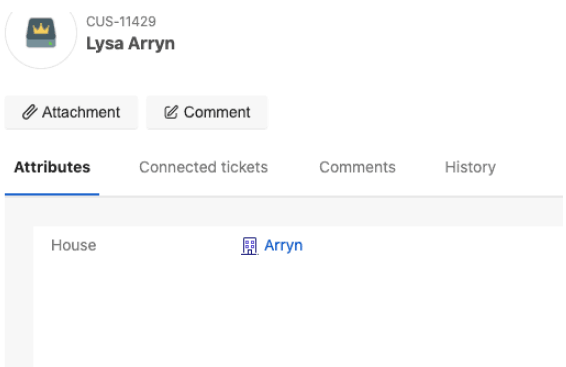
Team Blue

- Jira Service Desk with one service desk project to track all the details of the Game of Thrones series
- Insight asset management

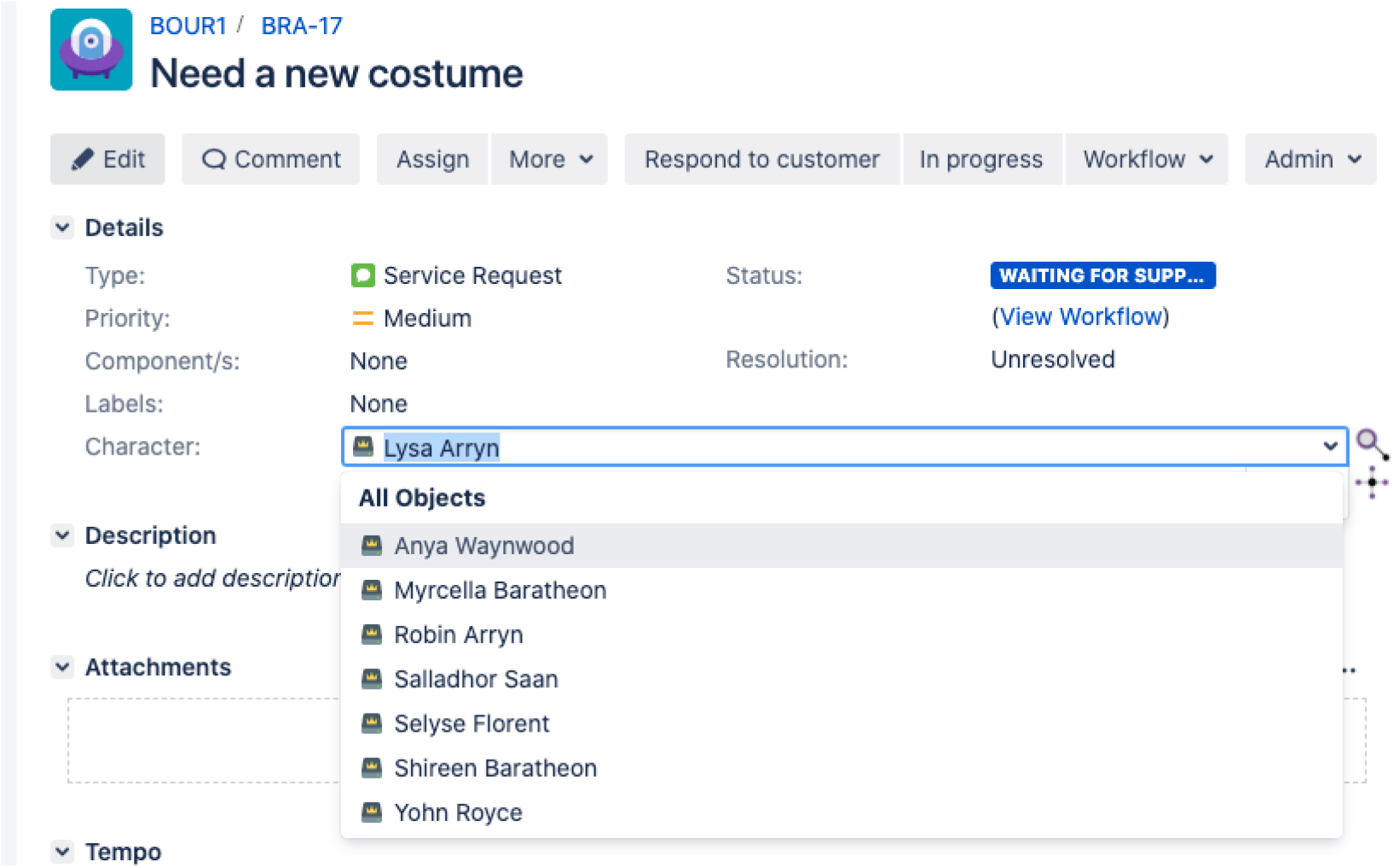
- Top-level object - GOT House
- One child - Character

T	Key	Name	House
	CUS-11426	Robin Arryn	Arryn
	CUS-11427	Yohn Royce	Arryn
	CUS-11428	Anya Waynwood	Arryn
	CUS-11429	Lysa Arryn	Arryn
	CUS-11430	Myrcella Baratheon	Baratheon
	CUS-11431	Selyse Florent	Baratheon
	CUS-11432	Shireen Baratheon	Baratheon
	CUS-11433	Salladhor Saan	Baratheon

The character has one attribute, 'House'.

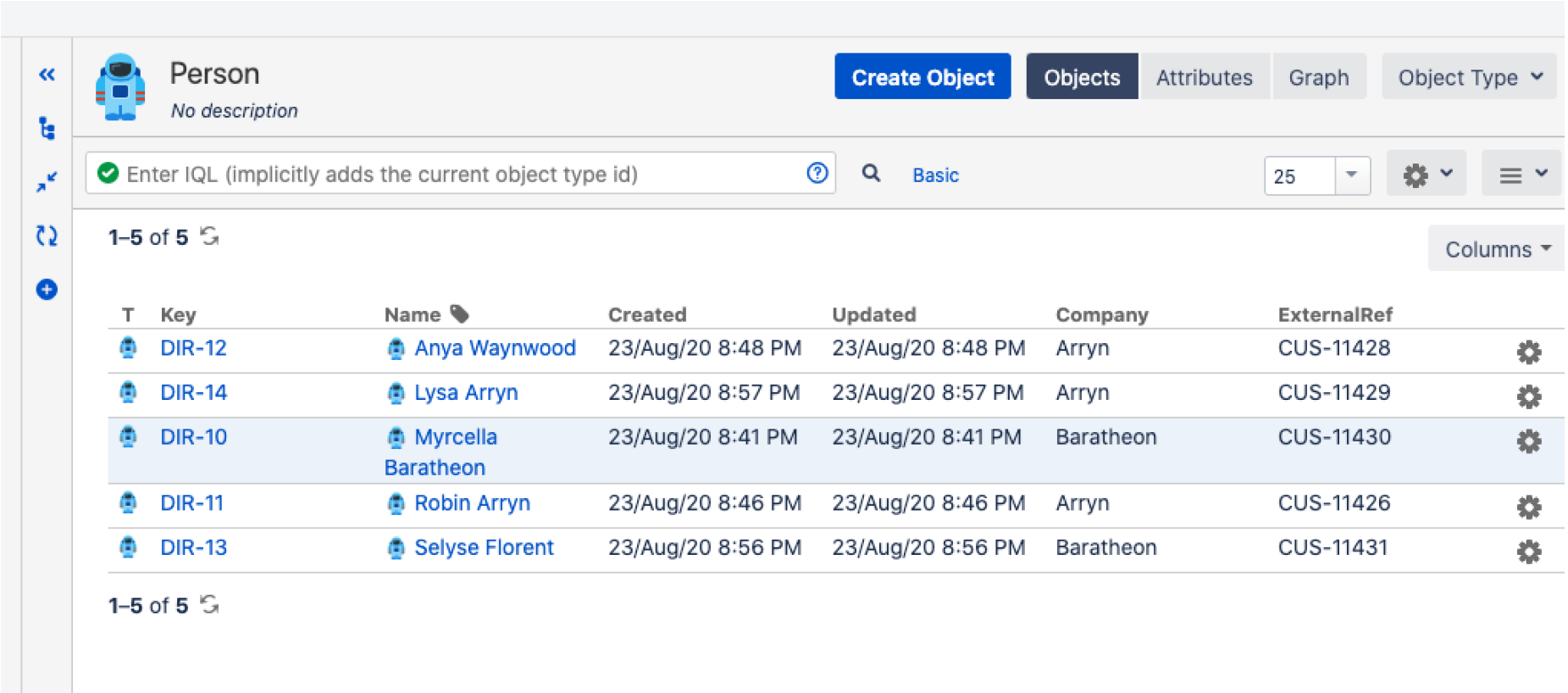


The ticket has an insight object custom field allowing to select from the drop-down.



Team Green

- One Jira Software project
- One Insight Object Schema, 'Directory'
- One Object type, 'Person'



Next to standard fields, one extra field has been added ('ExternalRef') which will be used to track the source of the person.

And it works!

Here is a video to demonstrate how the integration works:

Technical Details

Background Information

Most of the background information has been extracted from the Groovy Script Examples.

The point is that Exalate allows us to instantiate any Jira Service, even when it is delivered by an add-on such as Insight.
An example of how one can access Tempo Worklog attributes is available here. It boils down to:

```
// get the class of the service you would like to instantiate
def IOFacadeClass = ComponentAccessor.pluginAccessor.classLoader.findClass
("com.riadalabs.jira.plugins.insight.channel.external.api.facade.ObjectFacade")
// get the service
def IOFacade = ComponentAccessor.getOSGiComponentInstanceOfType(IOFacadeClass)
```

Overview of the Implementation

The approach for configuring an integration is always the same:

- On the source side (Blue), ensure that all required information is sent.

- On the destination side (Green), apply the information.

For this use case:

- On Blue

  send the character name, character key, house name, and house key to the other side.
  The character key will be used to find the character in the directory by doing a query on the ExternalRef.

- On Green
  - Try to find the character using the character key.
    - If not found, create a new person.
  - Assign the target person to the custom key.

Details of the Implementation

Accessing the Insight app Services

```
// do not forget to import the ComponentAccessor
import com.atlassian.jira.component.ComponentAccessor

// Get the Insight Object Facade
def IOFacadeClass = ComponentAccessor.pluginAccessor.classLoader.findClass("com.riadalabs.jira.plugins.insigh
def IOFacade = ComponentAccessor.getOSGiComponentInstanceOfType(IOFacadeClass)

// Get the Insight Query Language for doing searches
def IQLFacadeClass = ComponentAccessor.pluginAccessor.classLoader.findClass("com.riadalabs.jira.plugins.insig
def IQLFacade = ComponentAccessor.getOSGiComponentInstanceOfType(IQLFacadeClass)
```

```
// Get Insight Object Type Facade from plugin accessor
Class IOTypeClass = ComponentAccessor.getPluginAccessor().getClassLoader().findClass("com.riadalabs.jira.plug
def IOType = ComponentAccessor.getOSGiComponentInstanceOfType(IOTypeClass);

// Get Insight Object Attribute Facade from plugin accessor
Class IOTypeAttClass = ComponentAccessor.getPluginAccessor().getClassLoader().findClass("com.riadalabs.jira.p
def IOTypeAtt = ComponentAccessor.getOSGiComponentInstanceOfType(IOTypeAttClass);


// Get Insight Object Attribute Bean Factory
Class IOAttBeanClass = ComponentAccessor.getPluginAccessor().getClassLoader().findClass("com.riadalabs.jira.p
def IOAttBean = ComponentAccessor.getOSGiComponentInstanceOfType(IOAttBeanClass);
```

On source - extract attributes from a specific object and add to the replica.

```
// This code section goes into your outgoing sync processor of the connection

// Character is the Insight Object custom field, and we assume that the character has been provided

def CHARACTERNAME=824  // this id can be looked up in the Object Schema configuration
def CHARACTERHOUSE=827

def character = issue.customFields.Character?.value?.get(0)
def characterName = insightObjectFacade.loadObjectAttributeBean(character.getId(), CHARACTERNAME).getObjectAt

// dereference the house attribute
def characterHouse = insightObjectFacade.loadObjectAttributeBean(character.getId(),CHARACTERHOUSE).getObjectA
def refHouse = insightObjectFacade.loadObjectBean(characterHouse.getValue() as Integer)  // characterHouse.va
```

```
// add the attributes to the replica.customKeys such that these values are included in the message from blue

replica.customKeys.characterName = character?.name
replica.customKeys.characterKey = character?.objectKey
replica.customKeys.houseName = refHouse?.name
replica.customKeys.houseKey = refHouse?.objectKey
```

On target - search for the directory entry using the character key.

Note: *Searching/ creating the Insight objects requires an authenticated environment, which can be set up as documented here.*

```
// some constants
def DIRECTORYSCHEMA=1
def PERSONOBJECTTYPE=2
def PERSONNAMEATTRIBUTE=8
def PERSONCOMPANYATTRIBUTE=11
def PERSONEXTREFATTRIBUTE=12

// this goes in the incoming processor on green

def authContext = ComponentAccessor.getJiraAuthenticationContext()
def currentUser = authContext.getLoggedInUser()
def userKeyToSet = "admin"

def targetPerson
...

try {
    // ensure that the processor is run under a user permitted to search / create Insight objects
    // The try will catch will ensure that the user is returned to the current user
```

```
    authContext.setLoggedInUser(ComponentAccessor.getUserManager().getUserByKey(userKeyToSet));

    // Search for the person using an IQL which looks like 'ExternalRef in ("CUS-12345")'
    def iqlQuery = "\"ExternalRef\" IN (\"${replica.customKeys.characterKey}\")"
    def persons = IQLFacade.findObjects(DIRECTORYSCHEMA, iqlQuery)

    if (persons[0] == null) {
       // create the person
       ...
    } else {
      targetPerson = persons[0]
    }
} finally {
    authContext.setLoggedInUser(currentUser);
}
```

On target - create the object if not found.

```
if (persons[0] == null) {
        // found no record, so lets add it

       def IOTypePerson = IOType.loadObjectTypeBean(PERSONOBJECTTYPE);

       /* Create a new person object */
       targetPerson = IOTypePerson.createMutableObjectBean();


       /* Set up the attribute list */
       def IOAttBeans = new ArrayList();

       /* Set the name of the person */
```

```
        def nameBean = IOTypeAtt.loadObjectTypeAttributeBean(PERSONNAMEATTRIBUTE);
        IOAttBeans.add(IOAttBean.createObjectAttributeBeanForObject(targetPerson,
nameBean, replica.customKeys.characterName));


        /* Set the company of the person */
        def companyBean = IOTypeAtt.loadObjectTypeAttributeBean(PERSONCOMPANYATTRIBUTE);
        IOAttBeans.add(IOAttBean.createObjectAttributeBeanForObject(targetPerson,
companyBean, replica.customKeys.houseName));


         /* Set the external reference of the customer */
         def extRefBean = IOTypeAtt.loadObjectTypeAttributeBean(PERSONEXTREFATTRIBUTE);
         IOAttBeans.add(IOAttBean.createObjectAttributeBeanForObject(targetPerson,
extRefBean, replica.customKeys.characterKey));


         /* Set all object attributes to the object */
        targetPerson.setObjectAttributeBeans(IOAttBeans);

        /* Store the object into Insight. The targetPerson will be updated with an
unique ID */
        try {
            targetPerson = IOFacade.storeObjectBean(targetPerson);
            log.warn("targetPerson: " + targetPerson);
         } catch (Exception e) {
            log.warn("Could not create issue due to :" + e.getMessage());
         }

    } else {
        targetPerson = persons[0]

    }
```

```
} finally {

    authContext.setLoggedInUser(currentUser);

} if (persons[0] == null) {
        // found no record, so lets add it

       def IOTypePerson = IOType.loadObjectTypeBean(PERSONOBJECTTYPE);

       /* Create a new person object */
       targetPerson = IOTypePerson.createMutableObjectBean();


       /* Set up the attribute list */
       def IOAttBeans = new ArrayList();

       /* Set the name of the person */
       def nameBean = IOTypeAtt.loadObjectTypeAttributeBean(PERSONNAMEATTRIBUTE);
       IOAttBeans.add(IOAttBean.createObjectAttributeBeanForObject(targetPerson, nameBean, replica.customKeys


       /* Set the company of the person */
       def companyBean = IOTypeAtt.loadObjectTypeAttributeBean(PERSONCOMPANYATTRIBUTE);
       IOAttBeans.add(IOAttBean.createObjectAttributeBeanForObject(targetPerson, companyBean, replica.customK


        /* Set the external reference of the customer */
        def extRefBean = IOTypeAtt.loadObjectTypeAttributeBean(PERSONEXTREFATTRIBUTE);
        IOAttBeans.add(IOAttBean.createObjectAttributeBeanForObject(targetPerson, extRefBean, replica.customK
```

```
        /* Set all object attributes to the object */
       targetPerson.setObjectAttributeBeans(IOAttBeans);

       /* Store the object into Insight. The targetPerson will be updated with an unique ID */
       try {
           targetPerson = IOFacade.storeObjectBean(targetPerson);
           log.warn("targetPerson: " + targetPerson);
        } catch (Exception e) {
           log.warn("Could not create issue due to :" + e.getMessage());
        }

    } else {
       targetPerson = persons[0]

    }


} finally {

    authContext.setLoggedInUser(currentUser);

}if (persons[0] == null) {
        // found no record, so lets add it

       def IOTypePerson = IOType.loadObjectTypeBean(PERSONOBJECTTYPE);

       /* Create a new person object */
       targetPerson = IOTypePerson.createMutableObjectBean();


       /* Set up the attribute list */
       def IOAttBeans = new ArrayList();
```

```
        /* Set the name of the person */
        def nameBean = IOTypeAtt.loadObjectTypeAttributeBean(PERSONNAMEATTRIBUTE);
        IOAttBeans.add(IOAttBean.createObjectAttributeBeanForObject(targetPerson,
nameBean, replica.customKeys.characterName));


        /* Set the company of the person */
        def companyBean = IOTypeAtt.loadObjectTypeAttributeBean(PERSONCOMPANYATTRIBUTE);
        IOAttBeans.add(IOAttBean.createObjectAttributeBeanForObject(targetPerson,
companyBean, replica.customKeys.houseName));


         /* Set the external reference of the customer */
         def extRefBean = IOTypeAtt.loadObjectTypeAttributeBean(PERSONEXTREFATTRIBUTE);
         IOAttBeans.add(IOAttBean.createObjectAttributeBeanForObject(targetPerson,
extRefBean, replica.customKeys.characterKey));


         /* Set all object attributes to the object */
        targetPerson.setObjectAttributeBeans(IOAttBeans);

        /* Store the object into Insight. The targetPerson will be updated with an
unique ID */
        try {
            targetPerson = IOFacade.storeObjectBean(targetPerson);
            log.warn("targetPerson: " + targetPerson);
         } catch (Exception e) {
            log.warn("Could not create issue due to :" + e.getMessage());
         }

    } else {
        targetPerson = persons[0]
```

```
    }
} finally {
    authContext.setLoggedInUser(currentUser);
}
```

On target - apply the found/ created person to the customField.

```
// The customField expects a collection
issue.customFields.Person.value = [targetPerson]
issue.customFields.Person.value = [targetPerson]
```

Additional functions that could be implemented:

- If the character changes the name, update the corresponding person.
- If the character changes the house or the house changes the name, update the corresponding company name.
- If the character is deleted, mark the person as deleted on the other side.